D1-82-0182

BOEING SCIENTIFIC RESEARCH LABORATORIES

Random Variables and Computers

George Marsaglia

Mathematics Research

May 1962

D1-82-0182

RANDOM VARIABLES AND COMPUTERS

by

George Marsaglia

Mathematical Note No. 260

Mathematics Research Laboratory

BOEING SCIENTIFIC RESEARCH LABORATORIES

May 1962

A paper given at the Third Prague Conference on Probability Theory in June, 1962. It will appear in the Proceedings of the Third Prague Conference.

RANDOM VARIABLES AND COMPUTERS

George Marsaglia
Seattle

0. Introduction and Summary

This paper is concerned with methods for generating, in a digital computer, sequences of numbers which will serve as realizations of a sequence of random variables. The problem has two parts - the problem of producing independent uniform random variables, and the problem of producing arbitrary random variables in terms of uniform random variables. Section 1 describes arithmetic procedures for generating uniform variables. It contains nothing new. Section 2 considers properties of linear transformations, modulo 1, of uniform variables. These are the theoretical analogues of the arithmetic procedures for generating uniform variables. The rest of the paper, sections 3 - 7, is devoted to the problem of producing arbitrary random variables in terms of uniform random variables.

1. Producing Uniform Random Variables.

The most widely used current method for generating uniform numbers is to successively generate the residues of some integer by a linear transformation. This was first suggested by Lehmer, [1], and has become the standard procedure. As a simple example, in the ring of residues of 100, the linear transformation $11x + 7$ starting with, say $x = 54$, generates the sequence

$$54,1,18,5,62,89,86,53,\ldots$$

and the apparently haphazard succession of integers suggests that the sequence might well serve as a sequence of random digits. It is easy

to choose the modulus and the linear transformation so that a large number of residues can be generated without repetition, see [2]-[6] for details and further references. Rotenberg, [5], has given what is probably the most suitable method for digital computers: To generate a sequence of residues $x_1, x_2, \ldots$ put

$$x_{i+1} = (10^k + 1)x_i + c, \qquad \text{modulo } 10^m$$

for a decimal computer, with $k \geq 2$ and c not divisible by 2 or 5, and

$$x_{i+1} = (2^k + 1)x_i + c \qquad \text{modulo } 2^m$$

for a binary computer, where $k \geq 2$ and c is odd. These transformations generate all of the possible residues for their particular modulus, and are particularly suitable for digital computers since no proper multiplications, only shifts and additions, are required.

As an example of how the procedure works in a computer, suppose we have a decimal machine which manipulates blocks of 8 digits. We might generate the residues of 10^8 systematically by the transformation

$$x_{i+1} = 10^3 x_i + x_i + 231 \pmod{10^8}$$

Starting with, say, 21329312, we would have

$$\begin{aligned} x_1 &= 21329312 \\ &\quad \underline{29312231} \\ x_2 &= 50641543 \\ &\quad \underline{41543231} \\ x_3 &= 92184774 \\ &\quad \underline{84774231} \\ x_4 &= 76959005 \\ &\quad \underline{59005231} \\ x_5 &= 35964236 \end{aligned}$$

Rather than viewing the x's as residues of 10^8, we would

think of them as fractions of 10^8, and thus the above sequence would provide us with

.21329312, .50641543, .92184774, .76959005, .35964336,...

as a realization of the sequence $u_1,u_2,u_3,\ldots$ of independent uniform [0,1] random variables. When the iterates are viewed as fractional parts of the modulus, our transformation takes the form

$$T(x) = 1001x + .00000231 \quad \text{(modulo 1)}$$

and this leads us to a consideration of the properties of such linear transformations.

2. Linear Transformations, Modulo 1, of a Uniform Variable.

Let u be a uniform [0,1] random variable, and suppose we produce a new random variable, completely dependent on u, by a linear transformation mod 1,

$$T(u) = nu + a \pmod 1 \qquad n \text{ an integer}, \quad 0 \leq a < 1.$$

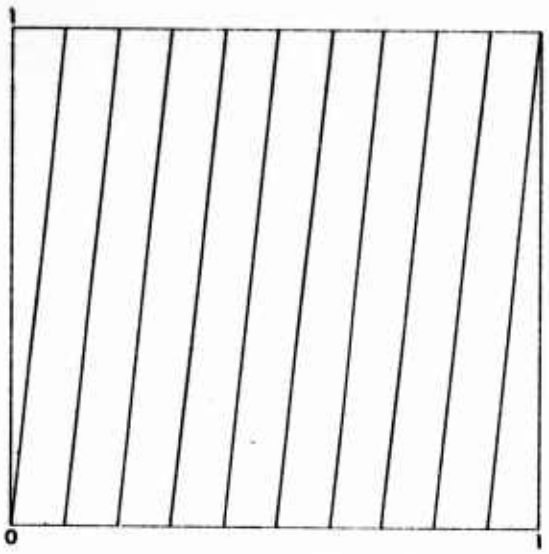
It is interesting to note that the sequence

$$(1) \qquad u, T(u), T^2(u), \ldots$$

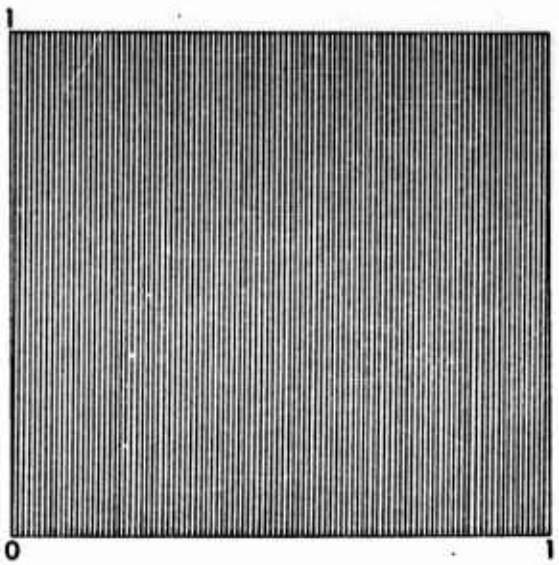
although a sequence of completely dependent variables, still has properties similar to those of a sequence $u_0,u_1,u_2,\ldots$ of independent uniform [0,1] random variables.

Suppose we look at the joint distribution of $u,T(u)$. They are individually uniform [0,1], but of course their joint distribution is degenerate. In fact, the point $(u,T(u))$ is uniformly distributed over the graph of T, and the larger n, the more the graph of T

tends to fill the unit square. In figure 1, graphs of T for $n = 10$ and 100 are drawn.



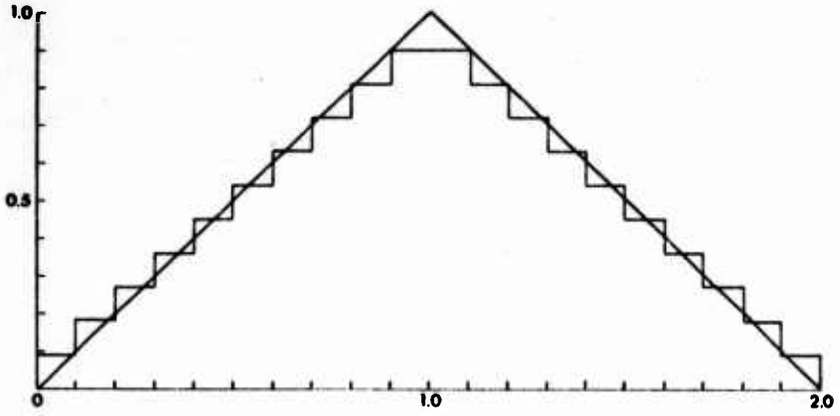
$T(u)=10u(\text{MOD } 1)$



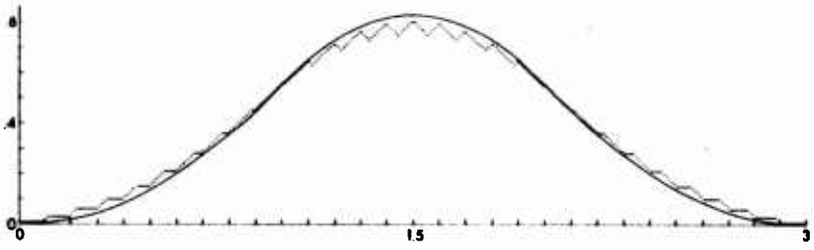
$T(u)=100u(\text{MOD } 1)$

Thus, for large n, the pair $u,T(u)$, while completely dependent, might well serve as a pair of independent uniform $[0,1]$ random variables.

It is easy to show that the sequence (1) satisfies the central limit theorem; in fact the sum $u + T(u) + \cdots + T^k(u)$ quickly converges to the distribution of the analogous sum $u_0 + u_1 + \cdots + u_k$ of independent uniform $[0,1]$ random variables. Figure 2 shows the density functions of $u + T(u)$ and $u + T(u) + T^2(u)$ compared with those of $u_0 + u_1$ and $u_0 + u_1 + u_2$, with $T(u) = 10u \pmod 1$.



Density function of $u+T(u)$ compared with that of u_0+u_1 .



Density function of $u+T(u)+T^2(u)$ compared with that of $u_0+u_1+u_2$.

The above considerations suggest that we should make n large in our arithmetic procedure for generating $u, T(u), T^2(u), \ldots$. Coveyou, [6], has suggested that T be chosen so as to reduce the correlation between u and $T(u)$. However, if n is large, that correlation will be negligible. We can easily find the correlation between u and $T(u)$. Suppose we use an overscore to indicate the fractional part of a number; thus $T(u) = \overline{a + nu}$ and in general, $\overline{x} = x - [x]$. Now we may write $u = \frac{d_1}{n} + \frac{v}{n}$ where d_1 is a random integer taking values $0,1,\ldots,n-1$ with probabilities $\frac{1}{n}$, and v is uniform $[0,1]$, independent of d_1. Then $T(u) = \overline{a + nu} = \overline{a + d_1 + v} = \overline{a + v}$. It is easy to verify that $\overline{a + v}$ is uniform $[0,1]$ and that the expected value of $v(\overline{a + v})$ is

$$E[v(\overline{a + v})] = \frac{2 + 3a(a - 1)}{6} .$$

Thus

$$nE[uT(u)] = E[(d_1+v)(\overline{a+v})] = E(d_1)E(\overline{a+v}) + E[v\overline{a+v}],$$

and it easily follows that the correlation, ρ, between u and $T(u)$ is

$$\rho = \frac{6(a - \frac{1}{2})^2 - \frac{1}{2}}{n}.$$

Thus $-\frac{1}{2n} < \rho < \frac{1}{n}$ and $\rho = 0$ when $a = \frac{1}{2} \pm \frac{1}{6}\sqrt{3}$. But in any case, ρ will be small when n is large.

3. Producing Arbitrary Random Variables

If we are willing to grant the adequacy of arithmetic procedures for generating uniform numbers, then we may assume we have a sequence

$$u_1, u_2, u_3, \ldots$$

of independent $[0,1]$ random variables and turn to the problem of producing arbitrary random variables in terms of the u's. An obvious way of generating a random variable x with continuous distribution F is to put $x = F^{-1}(u)$, but this is not usually feasible in a computer - it may take too long to evaluate $F^{-1}(u)$ in a subroutine, while a direct table look-up may require too many storage locations for the necessary precision. The storage requirement may be reduced by using a coarser mesh and interpolation, but even linear interpolation usually requires one multiplication, and we are concerned with procedures in which the time for a single multiplication would be a preponderant part of the average running time.

A general procedure which will provide fast programs for generating x with distribution F may be based on representing F as a mixture.

The method is described in [7], from which this summary is taken. The idea is roughly as follows: Suppose we have a method M_1 for providing a random variable y_1 with distribution function G_1, and method M_1 takes 10 units of time. Suppose we also have a method M_2 for providing a random variable y_2 with distribution function G_2, and method M_2 takes 500 units. If we can represent F as a mixture of G_1 and G_2, say

$$F(a) = .99G_1(a) + .01G_2(a),$$

then we may use $u_1,u_2,u_3,\ldots$ to provide a random variable x with distribution F as follows: If $u_1 < .99$, we use method M_1 on $u_2,u_3,\ldots$ to generate y_1 and put $x = y_1$. If $.99 \leq u_1 \leq 1$, we use method M_2 on $u_2,u_3,\ldots$ to generate y_2 and put $x = y_2$. The average running time will then be $(.99)10 + (.01)500 = 14.9$ units, plus the time necessary to test $u_1 < .99$.

This illustrates the basic principle of the simple device which provides programs with very short average running times - we represent F as a mixture of distributions,

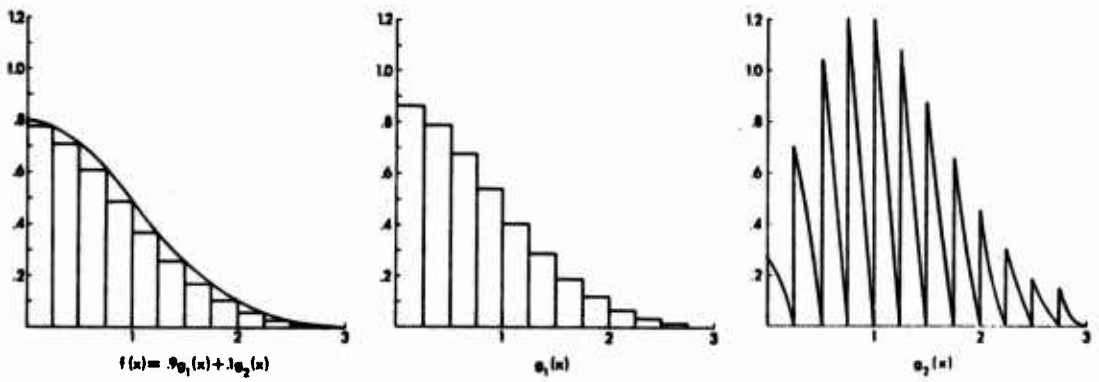
$$F(a) = p_1G_1(a) + p_2G_2(a), \tag{1}$$

in such a way that p_1 is close to 1 and the time to generate a random variable y_1 with distribution G_1 is small; then most of the time we put $x = y_1$. Even though G_2, the correcting distribution, may be quite complicated and difficult to handle, we still have a short average running time, since G_2 must be handled so infrequently.

In searching for representations of F such as (1), if we have a G_1 which shows promise, then we find the largest p_1 so that

$f(x) - p_1 g_1(x)$ is non-negative. Furthermore, if we make g_1 a rectangle or a mixture of rectangles, we can expect to have very short running times, especially if the rectangles are chosen so as to exploit the particular features of the computer in question.

References [7],[8] give details of how this method may be applied. The general idea is to break the density function into two parts, rectangles and "teeth", as for example, in this figure:

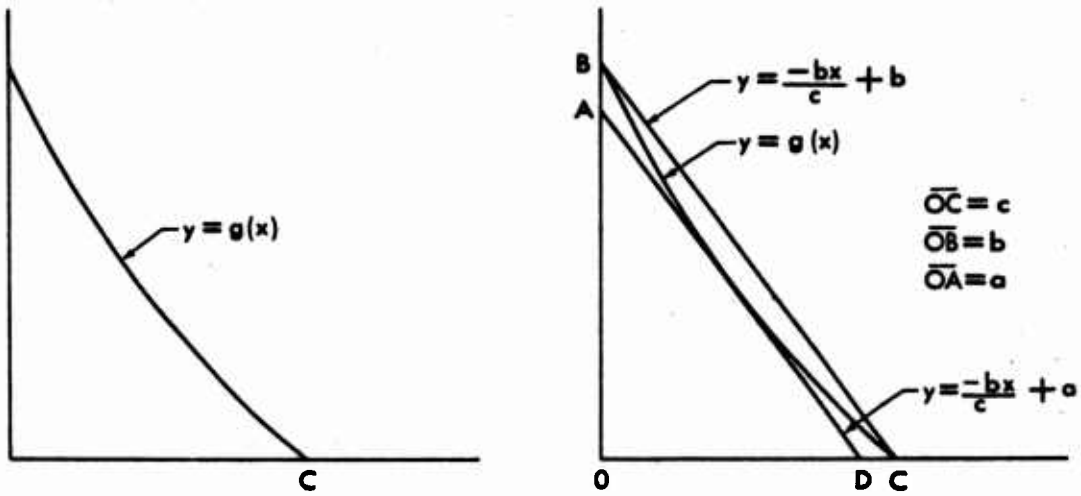


Then a random variable with density g_1 may be produced quite rapidly; only occasionally will the computer have to deal with the density g_2, and even then, a reasonable procedure may be constructed along these lines: One of the teeth of g_2 is chosen with appropriate probability, then a random variable with the nearly-linear density function represented by that tooth must be generated. We will describe a modified rejection

technique for generating such a variable in the next section.

4. Generating a Random Variable With a Nearly Linear Density.

Suppose we want to generate a random variable z with a nearly linear density function $g(x)$, $0 \leq x \leq c$, such as in this figure:



Let AD and BC be parallel segments enclosing $g(x)$. Then z may be produced as follows:

1) Choose independent uniform [0,1] random variables u and v, let $M = \max(u,v)$, $m = \min(u,v)$.

2) Choose a point (x,y) uniformly from triangle OBC, for example, by putting*

$x = cm$

$y = bM - bm$

3) If $y \leq -bx/c + a$, i.e., if (x,y) is in triangle OAD, put $z = x$ and stop.

4) If $-bx/c + a < y < g(x)$, put $z = x$ and stop.

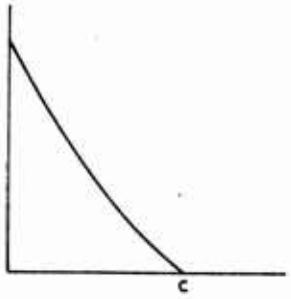
5) If $g(x) < y$, return to step 1 and try a new pair u,v.

*In fact, any one of the six representations (c−cM,bm),(cm,b−bM), (cM−cm,bm),(cm,bM−bm),(c−cM,bM−bm),(cM−cm,b−bm) will provide a point (x,y) uniformly distributed over triangle OBC. See, e.g., [9].

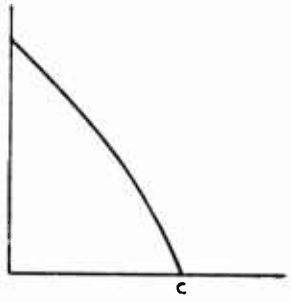
If we can enclose $g(x)$ in a narrow enough strip, we will produce x most of the time without computing $g(x)$ - step 3 will provide x with probability near 1. Note that the test in 3), $y \leq -bx/c + a$, is equivalent to $bM - bm \leq -bm + a$, or $M \leq a/b$, and a/b should be close to 1 if $g(x)$ is nearly linear. We may summarize the procedure as follows:

To generate a random variable z with a nearly linear density function $g(x)$, $0 \leq x \leq c$,

such as this

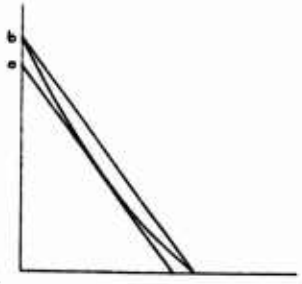


or this

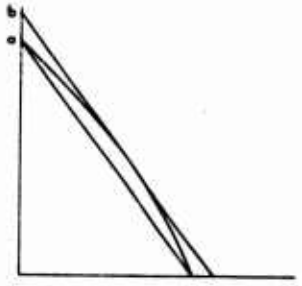


enclose $g(x)$ within two parallel lines,

like this



or this



Then:

1. Choose independent uniform [0,1] random variables u and v.
2. If $\max(u,v) \leq a/b$ put $z = c \min(u,v)$ and stop.

3. If not, test: $b|u - v| \leq g[c \min(u,v)]$? if yes, put $z = c \min(u,v)$; if no, go to step 1 and try again.

Note that for g concave up, as in the left case above, we may put $b = g(0)$ and $a = bx_0/c + g(x_0)$, where $g'(x_0) = -b/c$. If g is concave down, such as in the right case, then $a = g(0)$ and $b = -cg'(c)$ will do.

5. Special Methods for Particular Random Variables.

It is sometimes possible to take advantage of particular properties of the desired random variable in order to generate it. For example, as mentioned in [10], an exponential random variable X, density e^{-x}, $x > 0$, may be generated by putting

$$X = m + \min(u_1,u_2,\ldots,u_n)$$

where m and n are random integers with distributions:

$$P[m = k] = (e - 1)e^{k+1} \qquad k = 0,1,2,\ldots$$

$$P[n = k] = \frac{1}{k!(e - 1)} \qquad k = 1,2,3,\ldots$$

A possible method for generating a number of exponential random variables simultaneously is as follows: Let n be fixed, let $\pi = (p_1,p_2,\ldots,p_{n+1})$ be uniformly distributed over the regular simplex $S_{n+1} : \{(x_1,\ldots,x_{n+1}) : x_1+\cdots+x_{n+1}=1\}$ and let z have density $\frac{e^{-x}x^n}{n!}$, $x > 0$, and be independent of π. Then the $n+1$ random variables $zp_1,zp_2,\ldots,zp_{n+1}$ are mutually independent exponential random variables. A proof is in [9]. We may produce $(p_1,p_2,\ldots,p_{n+1})$ uniformly over S_{n+1} by ordering n independent uniform $[0,1]$ random variables $u_{(1)} \leq u_{(2)} \leq \cdots \leq u_{(n)}$,

then putting $p_1 = u_{(1)}, p_2 = u_{(2)} - u_{(1)}, p_3 = u_{(3)} - u_{(2)}, \cdots, p_n = u_{(n)} - u_{(n-1)}, p_{n+1} = 1 - u_{(n)}$.

A method for generating a pair (x,y) of normal random variables may be based on the polar representation, putting

$$x = \rho \cos \theta$$

$$y = \rho \sin \theta$$

where θ is uniform $[0,2\pi]$ and ρ is distributed as $\sqrt{x^2 + y^2}$, i.e., $\frac{\rho^2}{2}$ has the exponential distribution: $P[\rho < a] = 1 - e^{-a^2/2}$. As pointed out by Von Neumann, [11], $\cos \theta$ and $\sin \theta$ may be generated directly in the form $v_1/(v_1^2 + v_2^2)^{\frac{1}{2}}$, $v_2(v_1^2 + v_2^2)^{\frac{1}{2}}$ where v_1 and v_2 are uniform $[-1,1]$, conditioned by $v_1^2 + v_2^2 \leq 1$, or alternatively, avoiding the square root, in the form

$$\frac{2v_1v_2}{v_1^2 + v_2^2}, \quad \frac{v_1^2 - v_2^2}{v_1^2 + v_2^2}.$$

This procedure is quite slow for general use, but it does provide a convenient way of handling the tail of the normal distribution.

6. Generating a Discrete Random Variable.

In section 7 we will describe a general procedure for generating arbitrary random variables. It is based on the ability to generate a discrete random variable in a quick and simple manner. If x is a discrete random variable with $P[x = a_i] = p_i$, then an obvious method for generating x in a computer is to generate a uniform $[0,1]$ random variable u and put $x = a_i$ if

$$p_1 + \cdots + p_{i-1} < u \leq p_1 + \cdots + p_i.$$

Various search techniques based on this method, (see, e.g., [12]), may be used, but they often are relatively complicated programs that take too long. We will outline a procedure that leads to short and fast programs. To avoid notational difficulties, consider this particular example:

value of x	probability
a	.023 = .0+.02+.003
b	.038 = .0+.03+.008
c	.074 = .0+.07+.004
d	.103 = .1+.00+.003
e	.148 = .1+.04+.008
f	.206 = .2+.00+.006
g	.140 = .1+.04+.000
h	.101 = .1+.00+.001
i	.093 = .0+.09+.003
j	.037 = .0+.03+.007
k	.026 = .0+.02+.006
m	.011 = .0+.01+.001
	$\underset{.6}{.0}+\underset{.35}{.01}+\underset{.050}{.001}$

We have represented the probabilities in decimal form and will consider a decimal computer - the ideas apply equally well to a binary machine. Suppose our decimal computer has storage blocks (words) which may be called for by number, as most do. Then the fastest method for generating the discrete random variable x is probably this:

In memory locations 0-999, store 23 a's,38 b's,74 c's,103 d's, ···, 11 m's. Then if $u = .d_1d_2d_3d_4\cdots$ is a uniform random variable, generated in the computer in some way, look up the number in location $d_1d_2d_3$ and let that be x.

In effect, we have an urn containing 1000 balls - 23 marked a, 38 marked b, ··· and we choose one of the 1000 at random. While this is the fastest method, it requires 1000 storage locations. We will consider an improvement on the fastest method. It is an improvement

in that it uses much less memory space and takes little longer. To continue the urn analogy, suppose we fill 3 urns according to the digits of the probabilities for x as follows:

urn 1 - 1 d,1 e,2 f's,1 g,1 h (first digit)

urn 2 - 2 a's,3 b's,7 c's,...,1 m (second digit)

urn 3 - 3 a's,8 b's,4 c's,3 d's,···,1 m (third digit)

Now choose urn 1 with probability .600, urn 2 with probability .350, urn 3 with probability .050; then choose a ball at random from that urn. We then require only 91 balls instead of 1000, and,on the average, only a few more steps than the fastest urn scheme.

In a computer, we would "stack" the urns - urn 1 would occupy storage locations 0-5, urn 2 locations 6-40, and urn 3 locations 41-90. An outline of a program for generating this particular x might run:

Load constants in memory locations 0-90 according to this scheme:

Location Contents

0 d	10 b	20 e	30 i	40 m	50 b	60 e	70 f	80 j	90 m
1 e	11 c	21 e	31 i	41 a	51 b	61 e	71 f	81 j	91
2 f	12 c	22 g	32 i	42 a	52 c	62 e	72 f	82 j	92
3 f	13 c	23 g	33 i	43 a	53 c	63 e	73 h	83 j	93
4 g	14 c	24 g	34 i	44 b	54 c	64 e	74 i	84 k	94
5 h	15 c	25 g	35 j	45 b	55 c	65 e	75 i	85 k	95
6 a	16 c	26 i	36 j	46 b	56 d	66 e	76 i	86 k	96
7 a	17 c	27 i	37 j	47 b	57 d	67 f	77 j	87 k	97
8 b	18 e	28 i	38 k	48 b	58 d	68 f	78 j	88 k	98
9 b	19 e	29 i	39 k	49 b	59 e	69 f	79 j	89 k	99

Let $u = .d_1d_2d_3\cdots$ be a uniform [0,1] random variable. Let a(n)

be the contents of memory location n. Then

1) If $d_1 < 6$ put $x = a(d_1)$

2) If $60 \leq d_1d_2 < 95$ put $x = a(d_1d_2 - 54)$

3) If $950 \leq d_1d_2d_3$ put $x = a(d_1d_2d_3 - 919)$

Examples: $u = .217... \longrightarrow x = a(2) = f$

$u = .728... \longrightarrow x = a(18) = e$

$u = .963... \longrightarrow x = a(44) = b$

7. The Latest and Best Method

The ability to provide a discrete random variable quickly and easily serves as the basis for a general method for producing arbitrary random variables. This general method is very fast and easy to program for a computer. It is the latest and best of a series of procedures we have recently considered. The idea is quite simple - to produce a random variable y with distribution F, we write $y \cong x + v$, where x is discrete and v uniform. This amounts to replacing the distribution F by a piecewise-linear distribution G. By making x a random multiple of a small increment δ, we can make G as close to F as we please. Details of the procedure for the common distributions will appear elsewhere. We illustrate the procedure by showing how to generate an exponential variable: To generate y with $P[y < a] = F(a) = 1 - e^{-a}$, we generate y in the interval $0 \leq y \leq 4.6$ by putting $y = x + v$ where x takes values $0,.1,.2,...,4.5$ with

$$P[x = \frac{k}{10}] = p_k = e^{-\frac{k}{10}} - e^{-\frac{k+1}{10}} \qquad k = 0,1,2,...,45,$$

and v is uniform $[0,.1]$. Values of y in the interval $4.6 \leq y < \infty$

are produced by taking the logarithm of a uniform number.

The probabilities for x are expressed only to enough decimal places to give the accuracy necessary for the portion of F to which p_k applies, and the p's are rounded off, high or low, in such a way that the cumulative distribution $p_1 + p_2 + \cdots + p_k$ stays close to F. The p's are displayed in the table of constants for loading the memory; the digits of the p's are used to provide the compact storage method described in section 6.

The procedure below will produce a value y with distribution G for which $|F^{-1} - G^{-1}| \leq .001$, that is, it will return a value y which differs from the "true" value by less than .001. The steps of the procedure, very easy to program for a computer, are as follows:

To produce an exponential random variable Y, density e^{-y}, $y > 0$, store the constant $a(n)$, $n = 0,1,\ldots,569$ in memory location n. Let $u = .d_1d_2d_3\ldots$ be a uniform $[0,1]$ random variable, the d's its decimal digits. Then

1) If $0 \leq d_1d_2 < 79$ put $Y = a(d_1d_2) + .0d_6d_7d_8 \ldots$

2) If $790 \leq d_1d_2d_3 < 969$ put $Y = a(d_1d_2d_3 - 711) + .0d_6d_7d_8 \ldots$

3) If $9690 \leq d_1d_2d_3d_4 < 9888$ put $Y = a(d_1d_2d_3d_4 - 9432) + .0d_6d_7d_8 \ldots$

4) If $98880 \leq d_1d_2d_3d_4d_5 < 98994$ put $Y = a(d_1d_2d_3d_4d_5 - 98424) + .0d_6d_7d_8 \ldots$

5) If $98994 \leq d_1d_2d_3d_4d_5$ put $Y = -\ln(u - .98994)$

The constants for loading the memory are given in the table. The digits of the probabilities for x provide the frequencies of the

values to be stored. The second digit provides the frequencies for table 1, the third digit the frequencies for table 2, etc. The tables are then "stacked" in the memory.

Constants for Loading the Memory in Generating Exponential Random Variables

Value	Table 1	2	3	4
.0	.0	9	5	2
.1	.0	8	6	0
.2	.0	7	8	0
.3	.0	7	0	4
.4	.0	6	3	9
.5	.0	5	7	6
.6	.0	5	2	4
.7	.0	4	7	1
.8	.0	4	2	9
.9	.0	3	8	6
1.0	.0	3	5	1
1.1	.0	3	1	6
1.2	.0	2	8	7
1.3	.0	2	5	9
1.4	.0	2	3	5
1.5	.0	2	1	2
1.6	.0	1	9	3
1.7	.0	1	7	3
1.8	.0	1	5	8
1.9	.0	1	4	1
2.0	.0	1	3	0
2.1	.0	1	1	5
2.2	.0	1	0	7
2.3	.0	0	9	4 8
2.4	.0	0	8	6 4
2.5	.0	0	7	8 0
2.6	.0	0	7	0 8
2.7	.0	0	6	3 8
2.8	.0	0	5	8 0
2.9	.0	0	5	2 3
3.0	.0	0	4	7 5
3.1	.0	0	4	2 7
3.2	.0	0	3	8 9
3.3	.0	0	3	1 7
3.4	.0	0	3	5 2
3.5	.0	0	2	8 6
3.6	.0	0	2	6 1
3.7	.0	0	2	3 4
3.8	.0	0	2	1 4
3.9	.0	0	1	9 2
4.0	.0	0	1	7 5
4.1	.0	0	1	5 7
4.2	.0	0	1	4 4
4.3	.0	0	1	2 8
4.4	.0	0	1	1 8
4.5	.0	0	1	0 4

Table	Contents	Size of Table
1	nine 0's,eight .1's,seven .2's,...,one 2.2	79
2	five 0's,six .1's,eight .2's,...,one 4.5	179
3	two 0's,four .3's,nine .4's,...,one 4.4	198
4	eight 2.3's,four 2.4's,eight 2.6's,...,four 4.5's	114

Table	Memory Locations
1	0-78
2	79-257
3	258-455
4	456-569

In conclusion, it appears that, except for certain rare situations in which special tricks such as those in sections 3 or 5 may be more suitable, the best method for generating a random variable y is to generate a uniform number $u = .d_1d_2d_3\ldots$ by one of the arithmetic procedures, with enough digits in u that the first few may be used to assign a value to the discrete variable x by the compact storage method of section 6, then put $y = x + v$, where v is formed from the digits in the latter part of u. Occasionally, if u is close to 1, the tail of the distribution will provide y, rather than the simple procedure $y = x + v$. The compact storage procedure for providing x makes the method very easy to program, and the average time to produce y is little longer than the time to make a comparison, look up a stored value, and adjoin it to the last digits of u. Note that forming $x + v$ is not really an addition, but merely a juxtaposition of the digits of x and those of v.

The average time to produce u in one of the common U.S. computers, the IBM 7090, is about 30×10^{-6} seconds, and the average time to produce y from u is about the same, so that random variables $y_1, y_2, \ldots$ may be produced at the rate of about 17000 per second.

The status of the problem of generating random variables in a computer is then this: It is now fairly easy to develop a scheme for storing constants in a computer in such a way that an arbitrary random variable can be produced quickly and easily. The instructions for the computer are easy to formulate, and most of the time only basic operations such as testing magnitudes, shifting, and calling for stored values are

required. The procedures are easy enough that each computer center can develop standard subroutines of their own for providing the common random variables. Thus, mathematicians interested in using artificial sequences of random variables to guide their research activities can turn their attention to the problem of what to do with the large numbers of random variables that the computer can quickly and easily produce. Their first consideration will of course be to judge the suitability of the sequence for their particular need. In this connection, the traditional statistical tests on the random digits of the uniform numbers are of limited value. More appropriate is to devise a test problem similar to the one in question and try the sequence for that problem - if a problem on the random packing of spheres, try the sequence on a problem on the random packing of cubes for which the answer can be found. If one wants the distribution of a certain statistic for large values of the sample size n, try the artificial sequence on the largest n for which the distribution can be found by analytical means. If one wants to find the distribution of the length of a chain of molecules joined according to some chance mechanism, test the artificial sequence by generating a related random walk which has a known solution. If the problem involves some difficult function of order statistics, see if the artificial sequence gives results consistent with the distributions of some of the explored functions of order statistics, etc.

Mathematics Research Laboratory
Boeing Scientific Research Laboratories

REFERENCES

[1] D. H. LEHMER, "Mathematical Methods in Large-Scale Computing Units", Annals Comp. Laboratory Harvard Univ. 26, (1951) pp. 141-146.

[2] OLGA TAUSSKY and JOHN TODD, "Generation and Testing of Pseudo-Random Numbers", Symposium on Monte Carlo Methods, ed. Herbert A. Meyer, (Wiley, New York, 1956) pp. 15-28.

[3] A. DE MATTEIS and B. FALESCHINI, "Pseudo-Random Sequences of Equal Length", Comitato Nazionale per l'Energia Nucleare, Report No. 88, (Bologna, 1960).

[4] VACLAV DUPAC, "Metody Monte Carlo", Aplikace Matematiky, Sv. 7, C.1, (Praha, 1962) pp. 1-20.

[5] A. ROTENBERG, "A New Pseudo-Random Number Generator", J. Assoc. Comp. Mach. 7, (1960) pp. 75-77.

[6] R. R. COVEYOU, "Serial Correlation in the Generation of Pseudo-Random Numbers", J. Assoc. Comp. Mach. 7, (1960) pp. 72-74.

[7] G. MARSAGLIA, "Expressing a Random Variable in Terms of Uniform Random Variables", Annals Math. Stat. 32, (1961) pp. 894-898.

[8] G. MARSAGLIA, "Procedures for Generating Normal Random Variables, II", Boeing Scientific Research Laboratories, Mathematical Note No. 243, (Seattle, 1961).

[9] G. MARSAGLIA, "Uniform Distribution over a Simplex", Boeing Scientific Research Laboratories, Mathematical Note No. 250, (Seattle, 1961).

[10] G. MARSAGLIA, "Generating Exponential Random Variables", Annals Math. Stat. 32, (1961) pp. 899-900.

[11] JOHN VON NEUMANN, "Various Techniques Used in Connection with Random Digits", Monte Carlo Method, Nat. Bur. Stand., Appl. Math. Series 12, (1951) pp. 36-38.

[12] SETH ZIMMERMAN, "An Optimal Search Procedure", American Mathematical Monthly, Vol. 66, No. 8, (1959) pp. 690-693.